Using Timed Base-Choice Coverage Criterion for Testing Industrial Control Software

Henning Bergström and Eduard Enoiu
Mälardalen University, Västerås, Sweden

***Abstract*—The base-choice criterion was proposed as a suitable technique for testing software based on its nominal choice of input parameters. Test cases are created based on this strategy by varying the values of one input parameter at a time while keeping the values of the other parameters fixed on the base choice. However, this strategy might not be as effective when used on industrial control software for testing timed behavior. We propose to incorporate time as another parameter when generating and executing tests by defining the timed base-choice coverage criterion. We performed an empirical evaluation using 11 industrial programs written in the IEC 61131-3 programming language. We found that tests generated for timed base-choice criterion show better code coverage (7% improvement) and fault detection (27% improvement) in terms of mutation score than tests satisfying base-choice coverage criterion. The results demonstrate the feasibility of applying timed base-choice criterion for testing industrial control software.**

I. Introduction

Industrial control software [10] is used in real-time computers to provide operational process control of such systems as trains, car assembly lines and power plants. This type of software is used primarily for embedded systems, in which the behavior of the system depends not only on the choice of input parameters but also on the timing of the input parameters. For this kind of software, a test should not only provide the right choice of values, but these values should also be provided at the right time-points.

Combination test generation techniques are test generation methods where tests are created by combining the input values of the software based on a certain combinatorial strategy [5]. Base-choice [1] is a combinatorial technique used to generate tests by varying the values of one input parameter at a time while keeping the values of the other input parameters fixed to a base choice until all of the remaining combinations have been generated. However, this strategy might not be as effective when used on industrial control software because of its inherent limitation of just generating the right choice of values and not considering also the timing of the input parameters.

In this paper we study and extend the base-choice coverage criterion by including the time parameter for testing a special class of software known as industrial control software. In particular, we focus on IEC 61131-3 [7], a popular programming language used in different control systems. We performed an experimental evaluation using industrial software programs by comparing tests generated for the timed base-choice criterion with tests created based on the base-choice criterion in terms of achieved code coverage and fault detection. The results of this evaluation show that timed base-choice tests are more effective in detecting injected faults and can achieve better code coverage than tests created solely based on the base-choice criterion. This needs to be further studied; we need to consider the extent to which timed base-choice criterion can be used in the development of reliable industrial control software.

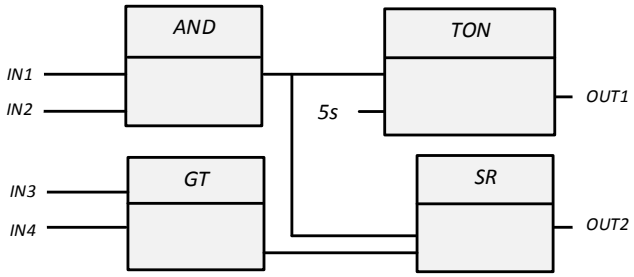
II. Background

In this section, major aspects of industrial control software and base choice coverage criteria are discussed. The presented aspects are related to the contribution of this study.

A. Industrial Control Software

An industrial control software is a type of software typically used in industries such as transportation, chemical, automotive, and aerospace to provide control and operation of critical infrastructures [10]. Programmable Logic Controllers (PLCs) are computer devices widely used for controlling industrial equipment. As an example of industrial control software, software running on a PLC [7] executes in a loop where every cycle contains the reading of input values, the execution of the program without interruption and the update of the outputs. These programs are written in one of the five languages defined by the International Electrotechnical Commission (IEC) within the IEC 61131-3 standard. An IEC 61131-3 program is invoked on a periodic basis. The periodic task scheduling is used by setting the cycle time P in time units as a positive integer number. For example in Figure 1a the body of the program is executed exactly once per cycle $P = 500ms$ in a loop that executes the program for as long as the PLC is working.

As shown in Figure 1a, blocks (e.g., SR, AND, GT, TON) and connections between blocks represent the behavior of an IEC 61131-3 program. These blocks are supplied by the hardware manufacturer or defined by the software developer. IEC 61131-3 programs contain blocks of a particular type, named timers (e.g., TON in Figure 1a). These timers are real-time instructions that provide the same functions as timing relays and are used to activate or deactivate a signal or a device after a preset interval of time T (i.e., $T = 5s$ in Figure 1a). A timer block keeps track of the number of times its input is either true or false and outputs different signals. From these basic timers many other timed behaviors can be created. Time in an IEC 61131-3 program is handled after the initial state by updating all the clock variables based on the time that has passed since the start of the program. For more details on this



(a) An example of an IEC 61131-3 program.

# Tests	IN1	IN2	IN3	IN4
1	1	1	5	4
2	0	1	5	4
3	1	0	5	4
4	1	1	5	3
5	1	1	5	5
6	1	1	4	4
7	1	1	3	4

(b) An example of a test suite satisfying base-choice criterion.

Fig. 1: A program with four inputs and two outputs written using the IEC 61131-3 programming language and a test suite satisfying base-choice criteria.

programming language we refer the reader to the work of John et al. [7].

B. Base-Choice Coverage Criterion

The base-choice coverage criterion was initially introduced by Ammann and Offutt [1]. This combination strategy works by varying the value of one input at the time while keeping all other inputs at fixed base values until all combinations have been used. This strategy causes each value choice of an input parameter to be used at least once together with the base choices of all the other input parameters. For example, consider the program in Figure 1a with four input parameters, where IN1 and IN2 have two value choices each (i.e., 0 and 1) while IN3 and IN4 have three value choices each (i.e., 3, 4, and 5). The program has a typical normal input behavior and this corresponds to a particular default choice (or base choice). Ammann and Offutt defined this default value choice as the value that corresponds to the normal model of operation for each input parameter. In the above example, we assume that $(1, 1, 5, 4)$ is the base test containing the default choice for each each input parameter. To satisfy base-choice coverage, for each value choice in an input parameter, we combine the specific choice with the base choice for all other input parameters. In the above example, if $(1, 1, 5, 4)$ is the base test, then we also choose additional tests shown in Figure 1b (i.e., Tests 2 to 7). As the program is executed exactly once per cycle P in a loop, tests are applied to the program every $500ms$. Industrial control software such as the one shown in Figure 1a contain timers that require input parameters to retain certain choice values for an amount of time in order to activate an event (i.e., OUT1 becomes active when the input of the TON block is true for more than $5s$). We note here that the tests shown in Figure 1b satisfying base-choice coverage are not adequate for activating OUT1. An engineer testing the above example program would want to add other useful tests.

Evidence on evaluating the effectiveness of base-choice criteria is sparse. Grindal et al. [5] showed that base-choice achieves high fault detection while using fewer tests than other combinatorial strategies when used on programs implemented in the C programming language. To our knowledge, no previous studies have looked at the use of base choice on industrial control software. This kindled our interest in studying how base-choice criterion can be effectively used for such systems.

III. Timed Base-Choice Strategy

As it turns out, the base-choice criterion as it is defined by Ammann and Offutt [1] is not particularly useful for IEC 61131-3 programs. Defining tests by selecting the value choices in each input parameter without taking into account the fact that the program is executed cyclically, results in an insufficient test suite (i.e., set of tests) for testing the normal operation of timers.

IEC 61131-3 programs typically have a time-depended behavior. This behavior require inputs to retain certain input values for some time in order to trigger events within the program. It is obvious that base-choice criterion in its original definition when used to generate tests for IEC 61131-3 software is not sufficient. We extend the base-choice coverage criterion by providing a time choice causing the varying input values to remain fixed for a certain amount of time T.

We define the timed base-choice criterion as follows: for each value choice in an input parameter, that choice is used in combination with the base choice for all the other input parameters for a certain amount of time determined by the time choice T. This causes each test satisfying base-choice criteria to be used several times for a certain amount of time. Some IEC 61131-3 programs might have multiple time choices T per program, but we consider only one normal time choice for simplicity in this study.

The following method creates tests that are based on the timed base-choice strategy for an IEC 61131-3 program:

1) *Create the input model.* The first step is to organize the input parameters in a list containing all the value choices. This is intended to be created automatically or manually by a test engineer.
2) *Identify the timing constraint.* This is typically related to the nominal timing behavior of the IEC 61131-3 program as taken from the functional requirements. The time choice should take into account that the values choices should be provided at the right time-points. In the example shown in Figure 1a such a timing requirement is illustrated: the output OUT1 must be activated after a certain amount of time. In this case, we can choose

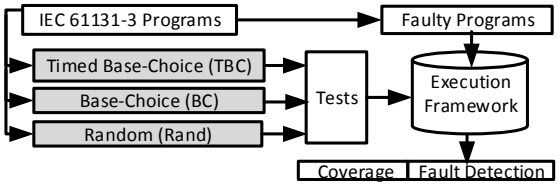


Fig. 2: Overview of the experimental method. For each program tests are generated and executed on both the original program as well as on the injected faults (i.e., mutated programs).

one time choice as a positive integer number measured in time units (e.g., $T = 6s$).

3) *Identify a base and time choice test.* For each input parameter, the test engineer chooses one of the value choices as the base choice. Ammann and Offutt [1] suggested that this can be typically a default or nominal value. The base choice test is obtained by selecting the base choice for each input parameter together with the time choice. In the above example, the base test $(1, 1, 5, 4)$ is fixed for $6s$.
4) *Create a test suite.* The input model together with the base and time choice can be used to automatically choose the remaining tests by varying over value choices for each input parameter and fixing these values for a certain amount of time (i.e., Tests 1 to 7 are fixed for $6s$ each).

This procedure can be used for creating executable test scripts for IEC 61131-3 programs for both base-choice and timed base-choice criteria.

The number of tests for timed base-choice criterion for a program cycle P, time choice T [1], N input variables where each input i has b_i choices is:

$$\frac{T}{P} + \sum_{i=1}^{N} \left((b_i - 1) \times \frac{T}{P} \right) \quad (1)$$

The number of tests needed for achieving timed base-choice is linear to the time choice T. Thus there is a significant execution cost disadvantage of using timed base-choice over base-choice.

IV. Case Study

We designed a case study for evaluating the use of base choice and timed base-choice criteria for testing IEC 61131-3 software. In this section we show how we selected the programs and performed the case study.

A. Method

The case study was performed according to the method shown in Figure 2. We aimed to answer the following research questions:

- *RQ1: Are timed base-choice tests able to detect more faults than random tests or base-choice tests?*
- *RQ2: Are timed base-choice tests able to achieve better code coverage than random tests or base-choice tests?*

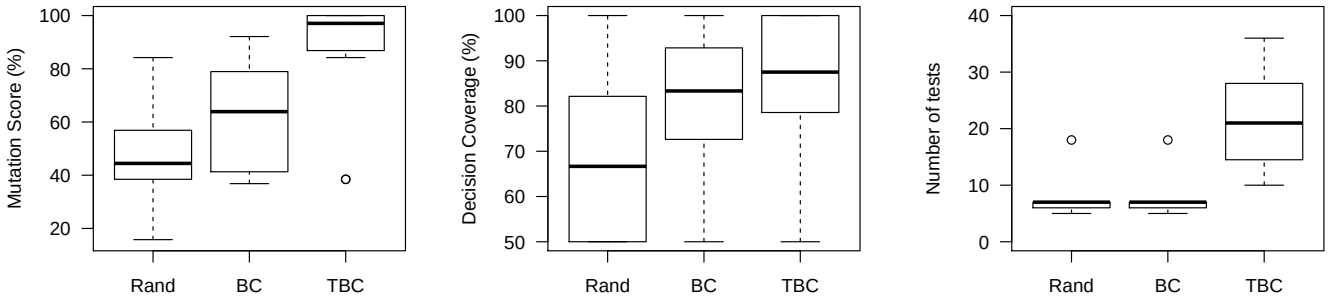
We started the case study by considering an industrial system actively developed by Bombardier Transportation, a large-scale company focusing on development and manufacturing of trains and railway equipment. The system is a train control management system (TCMS). TCMS is a distributed embedded system in charge of the operation-critical, safety-related functionality of the train. The system contains control and communication functions for high speed trains. These functions are developed as software programs using an IEC 61131-3 graphical programming language named Function Block Diagram (FBD) [6]. From a total of 20 IEC 61131-3 programs provided by Bombardier Transportation, we identified 11 candidate programs containing timers and timed behavior, making them suitable for comparing timed base-choice with base-choice criteria. We used the remaining programs for performing the case study. These programs contained on average per program: 22 decisions (i.e., branches) and 13 input variables. For each of the eleven programs, we collected test suites generated based on timed base-choice and base-choice criteria as well as random test suites with the same size as their base-choice counterparts. The input parameter values were obtained from the comments contained in each program. In addition, we asked one test engineer from Bombardier Transportation, responsible for testing the IEC 61131-3 programs used in this study, to carefully identify the base choice value of each input parameter. The test engineer provided base choice values for all input variables. The test engineer also provided a time choice to be used for generating timed base-choice tests.

In this study several measures were collected for answering the research questions. Code coverage criteria are used in software testing to assess the thoroughness of tests [2] and the extent to which the software structure has been exercised. In this study, we are using the decision coverage criterion (e.g., branch coverage). For the TCMS programs selected in this study, the EN 50128 safety standard [3] requires test engineers to create tests achieving high decision coverage. Fault detection was measured using mutation analysis. Mutation analysis is the technique of creating faulty implementations of a program (usually in an automated manner) for the purpose of examining the fault detection ability of a test suite [4]. Just et al. [8] provided compelling evidence that the mutation score is a fairly good proxy for real fault detection ability. In the creation of mutants we rely on previous studies that looked at commonly occurring faults in IEC 61131-3 software [9]. In total, for all of the selected programs, 867 mutants (faulty programs) were generated by automatically introducing faults into the original program.

B. Results and Discussion

This section provides an analysis of the data collected in this case study. The overall results of this study (i.e., mutation

[1] T and P are measured in the same time units



(a) Mutation score comparison (b) Decision coverage achieved (c) Number of tests in a test suite

Fig. 3: Mutation score, achieved code coverage and number of test comparison between timed base-choice tests (TBC), base-choice tests (BC) and pure random test suites (Rand) of the same size as the ones created for base-choice tests; boxes spans from 1st to 3rd quartile, black middle lines mark the median and the whiskers extend up to 1.5x the inter-quartile range and the circle symbols represent outliers.

scores, code coverage results and the number of test cases for each of the collected test suites) are summarized in the form of boxplots in Figure 3. To answer RQ1 regarding the effectiveness in terms of fault detection, we focused on comparing the test suite quality. For all programs, as shown in Figure 3a, the fault detection scores of TBC test suites are showing an average mutation score of 84% and are clearly superior to either BC test suites (57% mutation score on average) and random test suites (42% mutation score on average). Hence a test being created using TBC is a good indicator of test effectiveness in terms of fault detection. As seen in Figure 3b, the use of TBC achieves on average 85% decision coverage. The answer to question RQ2 regarding code coverage is matching our expectations: TBC test suites achieve higher code coverage than BC (on average 78%) or random test suites (on average 65%). Based on the results highlighted in Figure 3c, the use of TBC consistently results in significantly more number of tests (with 300% more tests on average) for all programs compared to BC.

The results show that timed base-choice criterion is useful for designing tests for industrial control software containing timing behavior. The definition of base-choice criterion has offered no information as to which tests to generate for properly testing the timed behavior in an industrial control software. In this paper we have defined the timed base-choice coverage criterion, and demonstrated the feasibility of applying this criterion.

V. Conclusions

In this study we proposed the use of timed base-choice criterion for testing IEC 61131-3 industrial control software. We conducted a case study in which we compared the cost and effectiveness between timed base-choice and base-choice created test suites. We used recently developed real-world industrial programs written in the IEC 61131-3 programming language, a popular language for developing safety-critical embedded systems using programmable logic controllers. The results show that timed base-choice generated test suites achieve better code coverage and fault detection compared to base-choice created test suites. The use of timed base-choice criterion needs to be further studied; we need to consider the implications of using multiple base and time choices. In addition, base-choice is only one type of combination strategy and we would need to evaluate the use of stronger criteria such as t-wise testing.

References

[1] Paul Ammann and Jeff Offutt. Using formal methods to derive test frames in category-partition testing. In *Computer Assurance, 1994. COMPASS'94 Safety, Reliability, Fault Tolerance, Concurrency and Real Time, Security. Proceedings of the Ninth Annual Conference on*, pages 69–79. IEEE, 1994.

[2] Paul Ammann and Jeff Offutt. *Introduction to Software Testing*. Cambridge University Press, 2008.

[3] CENELEC. 50128: Railway Application: Communications, Signaling and Processing Systems, Software For Railway Control and Protection Systems. In *Standard Official Document*. European Committee for Electrotechnical Standardization, 2001.

[4] Richard A DeMillo, Richard J Lipton, and Frederick G Sayward. Hints on Test Data Selection: Help for the Practicing Programmer. In *Computer*, volume 11. IEEE, 1978.

[5] Mats Grindal, Birgitta Lindström, Jeff Offutt, and Sten F Andler. An evaluation of combination strategies for test case selection. *Empirical Software Engineering*, 11(4):583–611, 2006.

[6] IEC. International Standard on 61131-3 Programming Languages. In *Programmable Controllers*. IEC Library, 2014.

[7] Karl-Heinz John and Michael Tiegelkamp. *IEC 61131-3: Programming Industrial Automation Systems: Concepts and Programming Languages, Requirements for Programming Systems, Decision-Making Aids*. Springer, 2010.

[8] René Just, Darioush Jalali, Laura Inozemtseva, Michael D Ernst, Reid Holmes, and Gordon Fraser. Are Mutants a Valid Substitute for Real Faults in Software Testing? In *International Symposium on Foundations of Software Engineering*. ACM, 2014.

[9] Younju Oh, Junbeom Yoo, Sungdeok Cha, and Han Seong Son. Software Safety Analysis of Function Block Diagrams using Fault Trees. In *Reliability Engineering & System Safety*, volume 88. Elsevier, 2005.

[10] Keith Stouffer, Joe Falco, and Karen Scarfone. Guide to Industrial Control Systems (ICS) Security. In *Special Report*, volume 800, pages 16–16. NIST, 2011.